Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (**Currently Amended**) An apparatus for allowing a user to model ~~modeling~~ at least one aspect of a software artifact, said apparatus comprising a processor and a memory storing code accessible by the processor to provide extension types, each extension type comprising an ordered tuple of a plurality of element types, each of the element types corresponding to different class hierarchies.

2. (**Original**) The apparatus according to Claim 1, wherein each extension type comprises an extension or variation of element types.

3. (**Original**) The apparatus according to Claim 1, wherein said extension types are adapted to compose classes horizontally.

4. (**Original**) The apparatus according to Claim 1, wherein each extension type is adapted to masquerade as any associated element type.

5. (**Original**) The apparatus according to Claim 1, wherein each extension type is a subtype of its associated element types.

6. (**Original**) The apparatus according to Claim 1, wherein:

each extension type has a size corresponding to the number of elements associated with the extension type; and

given two extension types α and β , a sub-type relation $\alpha <: \beta$ is definable as follows:

$|\alpha| >= |\beta|$; and

$\alpha(0) <: \beta(0), \alpha(1) <: \beta(1), \ldots \alpha(|\beta|-1) <: \beta(|\beta|-1)$.

7. (**Original**) The apparatus according to Claim 1, wherein, with α being the extension type of a variable *p* and β being the runtime extension type of the object pointed by *p*, so that $\beta <: \alpha$:

a method dispatch *p.m* comprises starting at the element type $\beta(0)$ and walking up the class hierarchy of $\beta(0)$ to find the closest *m,* wherein if *m* is not defined in the class hierarchy of $\beta(0)$, then *m* is sought in the $\beta(1)$ class hierarchy and, if needed, in one or more iteratively successive class hierarchies, until found.

8. (**Original**) The apparatus according to Claim 1, wherein, with α being the extension type of a variable *p* and β being the runtime extension type of the object pointed by *p*, so that $\beta <: \alpha$:

a method dispatch *p*m* comprises, for each element type $\beta(i)$, in the order $i=0, \ldots, |\beta|-1$, walking up the class hierarchy of $\beta(i)$ to find the closest *m* in $\updownarrow(i)$ and dispatching

the method *m* (if found), whereby a type error arises if *m* is not defined in at least one of the class hierarchies $\updownarrow(i)$, $i=0, \ldots, |\beta|-1$.

9. (**Original**) The apparatus according to Claim 1, wherein, with α being the extension type of a variable *p* and β being the runtime extension type of the object pointed by *p*, so that $\beta<:\alpha$:

a method dispatch *p(1,3,4).m* comprises reviewing only a class hierarchy of $\updownarrow(1)$, $\updownarrow(3)$, and $\updownarrow$ (4)to find the closest *m,* wherein a type error arises if *m* is not defined in any of $\updownarrow$ (1), $\updownarrow$ (3), or $\updownarrow$ (4).

10. (**Original**) The apparatus according to Claim 1, wherein, with α being the extension type of a variable *p* and β being the runtime extension type of the object pointed by *p*, so that $\beta<:\alpha$:

a method dispatch *p(1,3,4)*m* comprises reviewing only a class hierarchy of $\updownarrow(1)$, $\updownarrow(3)$, and $\updownarrow$ (4)to find the closest *m* in $\updownarrow(i)$ and dispatching the method *m* if found, whereby a type error arises if in any of the class hierarchies to which $\updownarrow$ (1), $\updownarrow$ (3), or $\updownarrow$ (4) belongs *m* is not defined.

11. (**Currently Amended**) A computer implemented method <u>for allowing a user to model</u> ~~of modeling~~ at least one aspect of a software artifact, said method comprising the step of providing extension types, each extension type comprising an ordered tuple of a plurality of element types, each of the element types corresponding to different class

hierarchies, wherein said extension types are stored in a memory of at least one general-purpose computer.

12. (**Original**) The method according to Claim 11, wherein each extension type comprises an extension or variation of element types.

13. (**Original**) The method according to Claim 11, wherein the extension types are adapted to compose classes horizontally.

14. (**Original**) The method according to Claim 11, wherein each extension type is adapted to masquerade as any associated element type.

15. (**Original**) The method according to Claim 11, wherein each extension type is a subtype of its associated element types.

16. (**Original**) The method according to Claim 11, wherein:

each extension type has a size corresponding to the number of elements associated with the extension type; and

given two extension types α and β , a sub-type relation $\alpha <: \beta$ is definable as follows:

$|\alpha| >= |\beta|$; and

$\alpha(0) <: \beta(0), \alpha(1) <: \beta(1), \ldots \alpha(|\beta|-1) <: \beta(|\beta|-1)$.

17. (**Original**) The method according to Claim 11, wherein, with α being the extension type of a variable *p* and β being the runtime extension type of the object pointed by *p*, so that $\beta <: \alpha$:

a method dispatch *p.m* comprises starting at the element type $\beta(0)$ and walking up the class hierarchy of $\beta(0)$ to find the closest *m,* wherein if *m* is not defined in the class hierarchy of $\beta(0)$, then *m* is sought in the $\beta(1)$ class hierarchy and, if needed, in one or more iteratively successive class hierarchies, until found.

18. (**Original**) The method according to Claim 11, wherein, with α being the extension type of a variable *p* and β being the runtime extension type of the object pointed by *p*, so that $\beta <: \alpha$:

a method dispatch *p*m* comprises, for each element type $\beta(i)$, in the order $i=0, \ldots, |\beta|-1$, walking up the class hierarchy of $\beta(i)$ to find the closest *m* in $\updownarrow(i)$ and dispatching the method *m* (if found), whereby a type error arises if *m* is not defined in at least one of the class hierarchies $\updownarrow(i)$, $i=0, \ldots, |\beta|-1$.

19. (**Original**) The method according to Claim 11, wherein, with α being the extension type of a variable *p* and β being the runtime extension type of the object pointed by *p*, so that $\beta <: \alpha$:

a method dispatch *p(1,3,4).m* comprises reviewing only a class hierarchy of ↕(1), ↕(3), and ↕ (4)to find the closest *m,* wherein a type error arises if *m* is not defined in any of ↕ (1), ↕ (3), or ↕ (4).

20. (**Original**) The method according to Claim 11, wherein, with α being the extension type of a variable *p* and β being the runtime extension type of the object pointed by *p*, so that β<:α:

a method dispatch *p(1,3,4)*m* comprises reviewing only a class hierarchy of ↕(1), ↕(3), and ↕ (4)to find the closest *m* in ↕(*i*) and dispatching the method *m* if found, whereby a type error arises if in any of the class hierarchies to which ↕ (1), ↕ (3), or ↕ (4) belongs *m* is not defined.

21. (**Currently Amended**) A data storage device readable by machine, comprising a data structure stored on the device, the data structure being at least one extension type comprising an ordered tuple of a plurality of element types, each of the element types corresponding to different class hierarchies; wherein said at least one extension type allows a user to model at least one aspect of a software artifact.